(4)

unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NW-LIS-88-31-01	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Semiannual Technical Report No. 1 "VLSI Architectures & CAD"		5. TYPE OF REPORT & PERIOD COVERED Technical, Interim
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Northwest Laboratory for Integrated Systems		8. CONTRACT OR GRANT NUMBER(s) N00014-88-K-0453
9. PERFORMING ORGANIZATION NAME AND ADDRESS Northwest Laboratory for Integrated Systems University of Washington, Dept. of Computer Science, FR-35 Seattle, WA 98195		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS DARPA-ISTO 1400 Wilson Boulevard Arlington, VA 22209		12. REPORT DATE November, 1988
		13. NUMBER OF PAGES 31 pages
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research - ONR Information Systems Program - Code 1513: CAF 800 North Quincy Street Arlington, VA 22217		15. SECURITY CLASS. (of this report) unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

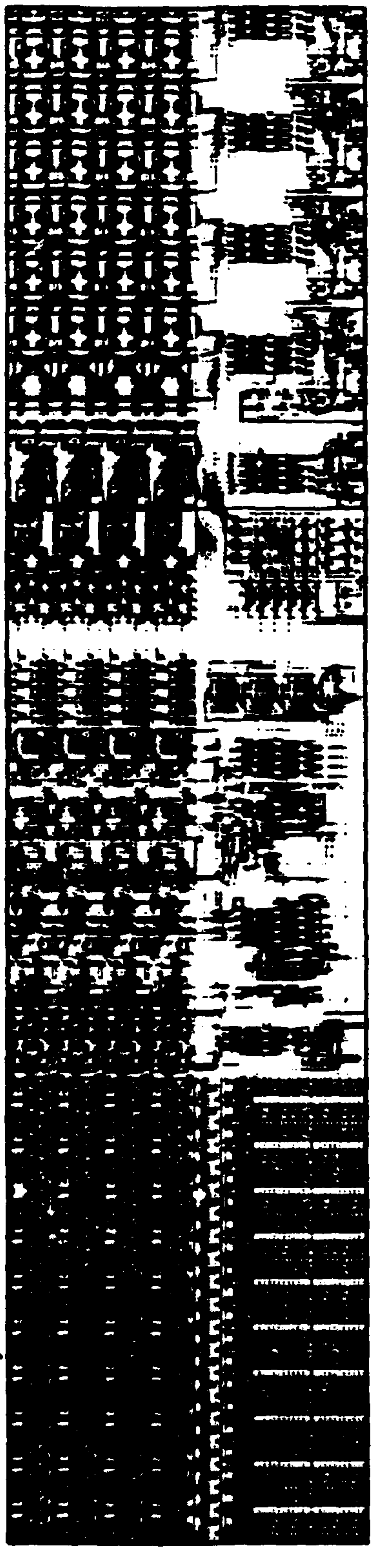
18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

NW LIS, VLSI, CMOS, APEX, Gemini, Wirelisp, Network C, CFL, RNL, parallelism, simulation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document reports on the research activities of the Northwest Laboratory for Integrated Systems, for the period of April 12, 1988, to November 14, 1988, under the sponsorship of the Defense Advanced Research Projects Agency and the Office of Naval Research, under contract number N00014-88-K-0453.



NORTHWEST LIS

(LABORATORY FOR INTEGRATED SYSTEMS)

Semiannual Technical Report No. 1
"VLSI Architectures & CAD"
University of Washington

November 14, 1988
TR #88-31-01

Reporting Period: April 12, 1988 - November 14, 1988

Principal Investigator: Lawrence Snyder

Sponsored by:
Defense Advanced Research Projects Agency (DoD)
Issued by Office of Naval Research
Under Contract #N00014-88-K-0453

Contents

1 Overview of Activities

Current research in the LIS is focused in several different areas. In the area of circuit specification is the development of *Wirelisp* - a dialect of *lisp* that facilitates the specification of circuit structure. A graphical frontend for *Wirelisp* allows the mixing of symbols representing circuit elements and *lisp* expressions.

Another area of continuing interest is the parallelism of CMOS circuits and the implications this parallelism has for simulation. The work has taken two forms. First, empirical studies have continued to provide insight into the potential parallelism of circuits, and to explain why designers feel that their circuits are more parallel than earlier measurements show. Secondly, we have begun to formally model different simulation strategies so that a comparison of the strategies can be made independent of the simulation implementation.

Current research is also being applied to the area of behavioral synthesis. A single representation is being sought that will describe both internal behavior (data-flow and operations) and interface behavior (signaling conventions and their timing constraints). A new unified behavior graph is being proposed that is concise and allows straightforward mapping to hardware (~~see Appendix~~ 1).

During the last six months several major design tools have been completed. They are being used extensively within the LIS and recently have been offered to the research community through the LIS distribution of VLSI design tools. These tools include *Network C* and *Gemini*, as well as *CFL* and *WIN*.

Network C is a multilevel simulation system that supports event-driven modeling at the functional and gate levels as well as continuous analog modeling. The event driven capability has been used with some success in the design of the APEX II chip; the analog capability has been used more recently in the design of the fuzzy CAM chip.

Gemini is a circuit comparison program that is used to compare netlists. Recently it has been modified to make it faster, enable it to isolate errors better and extend its domain of application. This revised version has been used extensively in the verification of the APEX I chip.

Several systems are now available for the development of design generators. One system, known as *CFL* (for Coordinate Free LAP), is a library of C routines for the assembly of layouts from leaf cells. Another system, known as *WIN* (for Washington Intermediate Notation), is a special purpose language which allows the parametric description of both geometric and electrical network information about circuit families.

2 APEX: An Architecture for Drawing Parametric Curves and Surfaces

(Tony DeRose, Mary Bailey, Bill Barnard, Robert Cypher, Carl Ebeling, Smaragda Konstantinidou, Larry McMurchie, Bill Yost)

APEX is an architecture that facilitates efficient drawing of curves and surfaces from sets of control points (see Appendix B, of LIS Semiannual TR, November, 1987). Two CMOS chips have been designed for implementing this architecture.

APEX I employs multiple processing elements in a triangular data-flow architecture. The chip contains about 65,000 transistors and was fabricated in an 84-pin frame using a 2-micron pwell process. We predict a clock rate of about 10Mhz. The design was extensively simulated using *RNL* and the layout validated using *Gemini*. Chips were submitted for fabrication in March, 1988 and were returned October, 1988. Testing of these chips is currently underway.

APEX II performs the same computation in a more flexible way, permitting the generation of a wider family of curve types of higher degree at the cost of lower performance. APEX II contains approximately 60,000 devices set in an 84-pin frame. It was fabricated using MOSIS' two-micron pwell process. Testing has been completed. All 6 major modules on the chip were tested individually and exhibited partial functionality. Three modules (two single-ported RAMs and a ROM) outputted data values with the wrong sense. Another module, a dual-ported RAM, gave inconsistent results under test; close examination of transistor ratios revealed a sensitivity of the design to process variation.

Problems with the RAMs pointed to weaknesses in the simulation strategy we employed. Our methodology consisted of using *RNL* to simulate a single module at a switch level while the remaining modules were described functionally using the *Network C* simulation system. In actual practice this technique was implemented using a UNIX fork of an *RNL* process from a *Network C* model. The technique worked very well for most modules on the chip and helped us isolate several problems. Unfortunately the RAMs and ROM could not be simulated with *RNL* – we had to rely on the functional models as adequately describing those circuits. Unfortunately the layouts diverged from the specification used to construct the functional models and produced the discrepancy in the sense of the outputs.

We have recently employed *COSMOS* with considerable success for the switch-level simulation of the modules in APEX II. By adjusting transistor strengths and directionality, we have been able to simulate all the modules in APEX II with *COSMOS*.

We have implemented the same UNIX forking scheme to simulate individual modules with *COSMOS* while the rest of the chip was being functionally simulated in *Network C*. All of the RAM/ROM problems were found using this strategy.

These problems were corrected and a successful switch-level simulation of the entire chip performed with *COSMOS*. The modified design is currently being fabricated.

3 Wirelisp: Schematics with Parentheses

(Carl Ebeling and Zhanbing Wu)

Wirelisp is a language used to describe the structure of a circuit. The structure can be described both graphically and procedurally where *lisp* expressions may be included in circuit drawings and circuit drawings may be included in *lisp* expressions. A number of features such as structured signals, iterators and optional parameters make the language very expressive. We have completed the first version of a *Wirelisp* environment. This includes a graphical programming interface for drawing *Wirelisp* programs, a backend analyzer which converts these drawings into *Wirelisp*, and a *Wirelisp* interpreter written in *T*.

We are currently defining extensions to *Wirelisp* to be implemented as user-defined functions. These will allow behavioral descriptions to be referenced within the structural description and allow physical information to be included as a means of driving a backend chip assembler. Behavioral descriptions are represented as modules for which the input-output behavior is specified explicitly, for example using logic equations or finite state machines. These modules reference both a behavioral description and a method by which the structure implementing that behavior can be generated. This implementation will vary depending on the particular output desired. For example, the implementation may be a functional model for *COSMOS* or *Network C*, a multi-level circuit generated by the *MIS* tools or a PLA. Our goal is to provide a simple framework in which the designer can specify and evaluate a system at a level appropriate to the level of detail required.

4 Investigations Into Circuit Parallelism

(Mary Bailey, Larry Snyder)

We have continued to focus on the potential parallelism of CMOS VLSI circuits, and the parallelism available for exploitation in discrete, event-driven circuit simulation. The work has taken two forms. First, empirical studies have continued to provide insight into the potential parallelism of circuits, and to explain why designers feel that their circuits are more parallel than earlier measurements show. Secondly, we have begun to formally model different simulation strategies so that a comparison of the strategies can be made independent of the simulation implementation.

In Appendix II of the April '88 LIS Semiannual TR, we presented two metrics for measuring the parallelism of CMOS VLSI circuits – the Event metric and the Queue metric. We also reported measurements of several circuits using the Event metric. We have continued measuring the parallelism of circuits using the Event metric, and have also measured the parallelism of these circuits using the Queue metric. The parallelism found using the Queue metric was much higher than that found using the Event metric. In some cases it was an order of magnitude higher! Appendix 3 of this report contains a discussion of the two metrics and a more complete comparison of these results.

In addition to considering the Queue metric, we have formulated a model for comparing the potential speedup of parallel simulators with different timing strategies. We have considered unit-delay, fixed-delay, and variable-delay synchronous strategies, and conservative asynchronous strategies. We found that as the timebase for variable-delay synchronous algorithms increases, the parallelism may increase, but never decreases. We also found that unit-delay provides as much parallelism as any of the synchronous algorithms, and that the conservative asynchronous algorithm sometimes provides more speedup than the unit-delay synchronous algorithm.

5 VLSI Design Tools, Release 3.2

We have recently begun distributing Release 3.2 of the LIS design system. This system contains tools written at the University of Washington as well as tools written at various other sites, including UC Berkeley, CMU and MIT. Major features of this system are described below.

Network C
(Bill Beckett)

Network C (*nc*) is a multilevel simulation system designed for constructing and simulating models of VLSI circuits and systems. The input language, a superset of C, supports a range of modeling capabilities including solution of Kirchoff equations at the analog level and discrete event functional simulation at the system level (Details of *nc*'s algorithms as well as experimental results have been described in LIS Semiannual TR's from March 1986 through April 1988).

Gemini
(Carl Ebeling)

Gemini is a circuit comparison program that is used to compare extracted circuit layout with a specification. Some recent extensions (See Appendix 2) make it faster, enable it to isolate errors better, and extend its domain of application. *Gemini*'s algorithm is separated into global labeling and local matching phases. *Gemini* dynamically switches between the two depending on the amount of local structure contained in the circuit, taking advantage of the speed of the local matching algorithm when possible, and relying on the power of the more general algorithm when the simple algorithm fails. This blending of algorithms also allows differences between two circuits to be better contained so that defects can be pinpointed.

Ohmics
(Wayne Winder)

Ohmics checks CMOS designs for the adequacy of ohmic contacts. The output of the circuit extractor *Mextra* is analyzed to determine the shortest path from each transistor to an ohmic contact of the correct type. The path is through p-well or p-substrate for n-channel devices and p-ohmic contacts, and though n-substrate or n-well for p-channel devices and n-ohmic contacts. *Ohmics* also determines that the ohmic contacts are electrically connected to the appropriate rail.

WIN
(Wayne Winder and Rudolf Nottrott)

WIN is a specialized circuit design language for assembling layouts and netlists (see Appendix A, LIS Semiannual TR, November, 1987).

CFL
(Bill Beckett)

CFL is a library of routines that allows parametrized layouts to be assembled within C programs (see Appendix I, LIS Semiannual TR March, 1986).

X11 support
(Warren Jessop)

We recently completed some *Magic* drivers for X Version 11. They are also available apart from the distribution through anonymous ftp on vlsi.cs.washington.edu. The compressed tar file is pub/magic11.tar.Z.

Portability considerations
(Warren Jessop)

The entire tools package now runs on 4 different hosts - SUN 3, DEC VAX (running ULTRIX), IBM RT (running Berkeley UNIX 4.3), and SEQUENT. As distributed, the tools package consists only of sources. "Make" procedures allow executables to be made for each of the target machines.

6 Progress on Fuzzy CAM Circuit Designs

(Bill Barnard)

Work has continued on content-addressable memories which match fuzzy or partially known data. Given a target input vector and a list of previously stored data vectors, these circuits return the minimum Hamming distance data vector in a single clock cycle. The additional circuitry for the fuzzy recall operation requires an additional 60% in area over a standard 4-transistor dynamic memory cell. We have chosen to call these circuits "fuzzy CAMs".

Previously, a 16 by 48 instance (16 vectors, 48 bits each) of a fuzzy CAM was fabricated through MOSIS. This fuzzy CAM was constructed using the layout assembly package *CFL*. The design was partially simulated with *RNL* (the analog nature of the circuit prevented a full *RNL* simulation) and partially simulated with SPICE (the full circuit being much too large to simulate completely). This chip was tested last summer, and was found to be only partially functional due to a design error in the dynamic memory refresh mechanism.

Our problems in the first design were caused by our inability (at that time) to simulate circuits with mixed analog and digital behavior. In a redesign effort, we have made extensive use of the *WIN* system for writing module generators as well as the mixed level simulation system *Network C.*

A fuzzy CAM layout/network generator was written using *WIN* notation. In addition to providing a tighter coupling between the layout and network descriptions, this recasting had the advantage of simplifying the generator code; source code for the *WIN* generator was a factor of 4 shorter than the original *CFL* generator.

The netlist output of the *WIN* generator can be used as input to *Network C* (*NC*). The goal is to simulate the fuzzy CAM in *NC*'s mixed mode framework and thus avoid the problems encountered in the first design, when some parts were simulated with *SPICE* and others with *RNL*. The analog fuzzy CAM circuitry has been simulated at an analog level while the peripheral circuitry and RAM has been simultaneously simulated at a gate level. So far this approach has been feasible only for relatively small instances of the fuzzy CAM, since the tightly coupled nature of the circuit limits the size that can be simulated.

In a parallel effort the entire fuzzy CAM has been simulated as a high-level functional model. The goal here is to use this model as a testbed for applications on real and semi-real data (e.g. digitized voice), and as a vehicle for developing tests of the fabricated chip. The idea is to use this functional model to simulate the processing of real data.

The Hamming distance evaluator part of the fuzzy CAM has been released for fabrication through MOSIS. Next steps include further simulation and eventual fabrication of an instance of the full fuzzy CAM design, further simulation of the high level functional model with real data, and evaluation of the new MOSIS double-poly double-metal technology for use in the analog circuitry.

ICCAD '88

APPENDIX 1

COMBINING EVENT and DATA-FLOW GRAPHS in BEHAVIORAL SYNTHESIS

Gaetano Borriello

Department of Computer Science, FR-35
University of Washington
Seattle, WA 98195

Abstract

The behavioral specification of a digital circuit consists of two parts: its internal behavior (data-flow and operations) and its interface behavior (signaling conventions and their timing constraints). Current behavioral synthesis systems use data-flow graphs to represent internal behavior. Recently, specialized synthesis systems have been developed that use event graphs to address the special nature of interface behavior. Combining event and data-flow graphs into a single unified representation has implications for how digital circuits are described and synthesized. This paper presents a new unified behavior graph and outlines the new algorithms required to support automatic synthesis. The new descriptive conventions are shown to be concise and to possess straightforward mappings to hardware. The algorithms are demonstrated to be of practical complexity ($O(n^2)$, where n is the number of interface events). A practical example demonstrates how the representation is used and synthesis results from five examples show that the synthesized circuitry is comparable to that achieved with other automatic methods or by experienced human designers.

1. Introduction

The description of a digital circuit consists of two parts, the internal and interface behaviors. Each part emphasizes different aspects of circuit operation. Internal behavior emphasizes data-flow and the operations that must be performed on the data. These consists of combinational and sequential logic elements that transform and store data and maintain circuit state. Interface behavior emphasizes the constraints imposed on the circuit by the environment in which it operates. These are the signaling conventions used for communication (i.e., the events or changes in logic state on signal wires) and the timing constraints between these events.

In commonly used hardware description languages (HDLs), these two aspects are mixed together. Actions related to the interface signaling conventions are embedded in the data-flow description although the two domains have quite different primitive elements. A more appropriate view of circuit description would include primitive elements from both domains. The internal specification includes only data-flow operations and the interface specification describes when inputs to the internal operators become available, when outputs must be generated, and the signaling conventions to be used with the circuit's environment. This approach permits the use of specialized synthesis methods for the two aspects of the circuit. Therefore, a unified graph representation of circuit behavior for general-purpose behavioral synthesis systems must include a clean interface between the two domains of description and methods for synthesizing the hardware components that will interconnect the interface to the internals.

This paper is composed of six sections. The first is this introduction to the issues to be considered. Section 2 summarizes related work and how it is inadequate for the task. Section 3 presents the model for a unified representation of internal and interface behavior. Sections 4 and 5 describe the implications of this representation for specification methods and synthesis algorithms, respectively. Finally, section 6 concludes the paper with some results and summary remarks.

2. Related Work

The different nature of the two behavioral domains has led to different representations for each. Historically, the emphasis has been on describing internal behavior. Data-flow graphs are in use as input to many behavioral synthesis tools that can generate complete designs under some cost and performance constraints [McFa88]. The nodes of these graphs represent combinational logic operations (e.g., comparisons and arithmetic) or access to internal state (e.g., memories and registers) while the arcs represent the data values being generated and used as inputs to the operations. Extensions for dealing with interface behavior have consisted of expressing constraints on the execution of a sequence of data-flow operations within the confines of a fully synchronous model of circuit behavior [Nest86]. However, this approach obscures data-flow with signaling operations. The description is difficult to write due to the potentially high level of concurrency between the two domains.

Work on self-timed circuits has lead to specification of interface behavior using Petri nets. Synthesis methods for asynchronous circuits have been developed based on this representation [Moln85, Chu86]. The need to represent and synthesize circuits with interface timing constraints has led to the development of event graphs

[Borr87]. The nodes of these graphs correspond to signaling events and the arcs specify how the events are ordered and separated in time. This model freely mixes synchronous and asynchronous interface behavior. However, only limited data-flow information is captured, namely, when input and output data values must be valid on the interface signal wires.

These two representations (i.e., data-flow and event graphs) are specialized for expressing behavior and synthesizing circuitry that falls within one or the other domain. The USC Design Data Structure seeks to unify the representation of these different behavioral domains by using two hierarchical graphs that roughly correspond to the internal and interface behavior domains [Knap85]. Bindings between elements in the two graphs are used to link the two types of information. Therefore, there must be a node in the data-flow graph for every event on the interface leading to large and complex graphs. A representation is needed with a more distinct separation between the two types of behavior and graph nodes and arcs with clear-cut mappings to hardware realizations. This is the approach being taken at UC Berkeley with OE-graphs, a behavior graph with two node types (operations and events) in a bi-partite arrangement [Sequ88]. Data-flow is observed by looking only at operation nodes and events by looking only a event nodes. However, interface signaling is again interspersed with data-flow operations.

3. A Unified Behavior Graph

The unified behavior graph presented in this paper is a hybrid of data-flow and events graphs that most closely resembles event graphs [Borr88]. The nodes correspond to operations, either signal events or data-flow operations; arcs correspond to either timing constraints (min/max) or data dependencies. Nodes include min/max durations: propagation delay for data-flow operations and rise and fall times for interface events. The interface portion of the graph consists primarily of event nodes and timing constraint arcs. The data-flow portion of the graph consists primarily of operation nodes and data dependency arcs.

Data arcs connect the interface behavior to the internal behavior. A data dependency arc from an input event to an operation signifies where and when the input data becomes available on the interface. A data arc from an operation to an output event signifies where and when output data is to be presented on the interface. Timing constraint arcs are propagated to the data-flow from the interface specification.

Conditional and iterative behavior is represented using the same solution adopted for event graphs [Borr88]. The graph is partitioned into segments and these are composed using a regular-expression syntax. Each segment must include a distinguished node that serves as the enabling condition for the segment. This can be a combination of events on interface signals or conditions on data. Both can be represented as nodes in the graph, the former by annotating the events and the latter by a comparison operation on the data. Iterations are similarly described with the difference that the enabling condition of the loop is re-tested after each iteration to determine whether the loop must be executed again. Arcs within an iterative segment must be distinguished as to whether they apply between nodes of the same or consecutive iterations.

4. Implications for Specification

The implications of the unified representation for behavioral description languages seem to all be beneficial. One of these is that the representation supports synchronous and asynchronous behavior equally well through the use of an asynchronous communicating processes model. Input data becomes available, is operated upon, and output data is made available. The use of specialized description methods for the interface and data-flow behavior is encouraged by the clean separation between the two portions of the graph. For example, an HDL with *send* and *receive* constructs can be used to describe the data-flow and timing diagrams can be used to represent the interface details [DeMi88, Borr88]. The interconnections between the two are made between the data event nodes on the interface and the send and receive operation nodes in the data-flow. Of course, the basic block structure of the description must be identical in the two representations.

An example of a two part circuit specification is shown in Figure 1. The circuit described accepts a byte-stream of data and outputs it again with a checksum byte appended to the end of the stream. An HDL is used to describe the data-flow and two timing diagrams describe the two parts of the circuit's interface. From the diagrams one can see that input bytes arrive asynchronously while the output bytes are generated synchronously. The unified graph corresponding to the description is shown in Figure 2.

Another advantage of the unified graph is that it enables interface constraints to propagate to the internal behavior. Data-path synthesis can include this information in its scheduling and allocation algorithms.

5. Implications for Synthesis

In the unified representation, there is no restriction on whether the data-flow circuitry is synchronous or asynchronous. This distinction can only be made for interface signals and then propagated to the internals of the circuit by high-level requirements on the cost and performance of the design. These requirements will determine which implementation style (e.g., synchronous, asynchronous, or self-timed) will be used. This is not the case with current behavioral synthesis systems that can only generate fully synchronous designs [McFa88].

Synthesis algorithms that run in polynomial time and yield near optimal results exist for both data-flow and interface circuitry [McFa88, Borr88]. The representation presented in this paper permits the use of these specialized synthesis methods on the two portions of the circuit's behavioral specification. The connections between the two parts must be shown to be synthesizable with comparable complexity to current methods for the unified representation to be viable.

The two new problems are with the inputs and outputs of the circuit. For input data that is to be transferred into the circuit, an interface event that signals the presence of the data must be identified and possibly used to control a latch. For output data, it must be determined if the circuit can generate it in time to satisfy interface timing constraints. Both of these problems can be solved using the concept of *input-ready/output-ready* events and a graph algorithm that determines intervals of occurrence for interface events.

An interval of occurrence can be determined for every node in the graph by fixing the position of one node and then traversing the arcs of the graph. The arcs and the durations of the nodes will constrain the interval during which a node can occur relative to the fixed node. The algorithm is similar to one used in layout compaction algorithms and is $O(em)$, where e is the number of edges and m is the number of maximum constraints [Burn86]. Typically, there is a small number of maximum constraints and the algorithm can be assumed to be $O(n)$ for each node requiring an input-ready event to be identified. This yields a worst-case complexity of $O(n^2)$.

The input-ready event is determined by fixing the event at which the data becomes valid and finding another non-data input event that occurs after the data event and before the data is deasserted. Alternatively, an event that occurs a specific amount of time before the data is asserted can be used (it can be delayed by the appropriate amount). This input-ready event is paired with the receive node at the beginning of a data-flow portion of the graph. The control for the data-flow circuit can either generate a synchronous pulse from the input-ready event or use it as a start signal for an asynchronous computation. Figure 2 explains which events are selected for the example graph.

The output-ready event is generated by the data-flow circuit and is used to enable the signaling events for the output data (see Figure 2). The interval of occurrence algorithm is used to determine the range of time within which this event must be generated and still satisfy the interface timing constraints. The output-ready event can either be an asynchronous completion signal or the synchronous output of a finite state machine controller.

6. Conclusions

A unified representation for circuit behavior has been presented and has been shown to have positive implications for behavioral specification and synthesis. The two behavioral domains of a circuit are linked using simple conventions in its HDL description that have direct mappings to hardware. The distinction between internal and interface behavior prevents data-flow details from being obscured by interface signaling considerations and interface circuitry can be synthesized independently of the data-flow circuitry. The effect on synthesis algorithms has been shown to be of practical ($O(n^2)$) complexity and circuitry has been synthesized for five examples with good results (see Table 1) [Borr88]. Furthermore, the clean separation of the interface from the internals permits the reuse of a data-flow specification with a different interface and the reuse of an interface specification with a different internal data-flow.

Circuit	Size	Performance
Counter	same	same
FIFO control cell	same	same
Multibus Design Frame	+17%	+9%
2-4 Phase Protocol Adapter	-39%	same
SPUR Cache Controller	-11%	same

Table 1. Synthesis results for five examples including fully synchronous, fully asynchronous, and mixed circuits. Size is measured in number of logic gates and performance in maximum communication bandwidth.

References

[Borr87] G. Borriello, R. Katz, Synthesis and Optimization of Interface Transducer Logic, IEEE International Conference on CAD, November 1987.

[Borr88] G. Borriello, A New Interface Specification Methodology and its Application to Transducer Synthesis, Technical Report UCB/CSD-88/430, Computer Science Division, University of California at Berkeley, May 1988.

[Burn86] J. Burns, A. Newton, SPARCS: A New Constraint-Based IC Symbolic Layout Spacer, Proceedings of the Custom Integrated Circuits Conference, 1986.

[Chu86] T. Chu, L. Glasser, Synthesis of Self-Timed Control Circuits from Graphs: An Example, IEEE International Conference on Computer Design, October 1986.

[DeMi88] G. DeMicheli, D. Ku, Hercules — A System for High-Level Synthesis, 25th Design Automation Conference, June 1988.

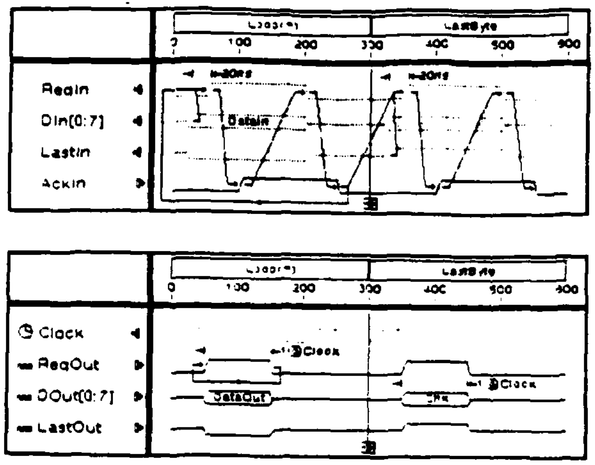
[Knap85] D. Knapp, A. Parker, A Data Structure for VLSI Synthesis and Verification, Technical Report CRI-85-19, Department of Electrical Engineering-Systems, University of Southern California, August 1985.

[McFa88] M. McFarland, A. Parker, R. Camposano, Tutorial on High-Level Synthesis, 25th Design Automation Conference, June 1988.

[Moln85] C. Molnar, T. Fang, F. Rosenberger, Synthesis of Delay-Insensitive Modules, 1985 Chapel Hill Conference on VLSI, May 1985.

[Nest86] J. Nestor, D. Thomas, Behavioral Synthesis with Interfaces, IEEE International Conference on CAD, November 1986.

[Sequ88] C. Sequin, personal communication, 1988.



```
circuit CheckSumGenerator;

    inputs   ReqIn, DIn[0:7], LastIn; Clock;
    outputs  AckIn, ReqOut, DOut[0:7], LastOut;
    clock    Clock;
    int      Byte[0:7], ChkSum[0:7];

    repeatbegin;
             receive(Byte, DataIn);
             cobegin;
                  send(Byte, DataOut);
                  ChkSum = Byte <op> ChkSum;
             coend;
    repeatend;
    send(ChkSum, Chk);

end CheckSumGenerator;
```

Figure 1. A specification for a circuit that takes a byte stream input, generates a checksum byte, and outputs the same stream with the checksum appended. The specification consists of three parts: a timing diagram for the input interface, a timing diagram for the output interface, and an HDL program for the data-flow portion. The input interface is asynchronous, the output interface is synchronous, and the internal data-flow is unspecified. Note that the basic block structure of the diagrams is identical to that of the program. The vertical lines in the diagrams separate it into segments and a regular expression is formed by the horizontal blocks above the traces of the diagram (a *Kleene star* is used for an unspecified number of iterations). The time line in the diagrams is for illustrative purposes only and is not part of the specification. The dotted arcs in the two diagrams identify constraints between consecutive iterations of the loop. The notation *1@Clock* in the second diagram is shorthand for a timing constraint that precisely spaces the two events one cycle of *Clock* apart.

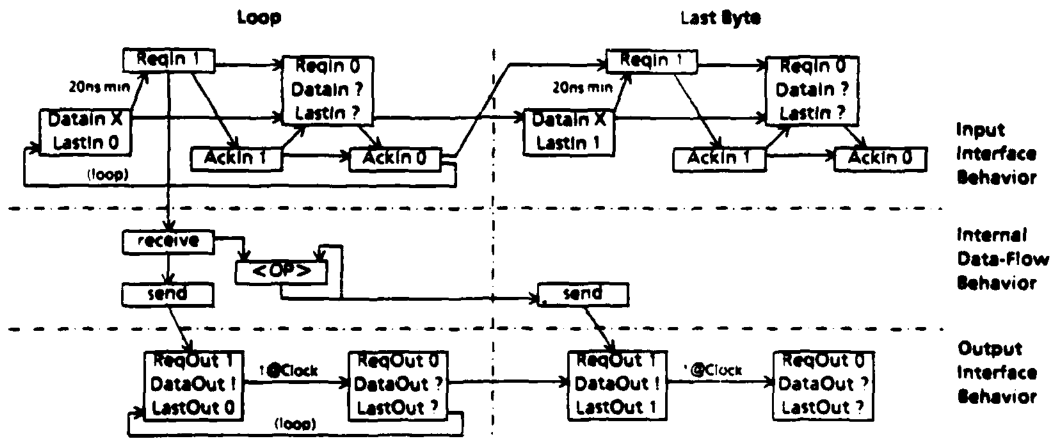


Figure 2. The graph representation of the example of Figure 1. The dashed vertical line breaks the graph into two parts corresponding to the basic blocks of the program and the segments of the diagrams. The two horizontal lines break the graph into parts corresponding to each of the two diagrams and the HDL program. Arcs in the interface behavior portions of the graph are timing constraint arcs while those at least partly within the internal behavior portion of the graph are data arcs. Each segment is enabled by the first event on *ReqIn*, the first segment is enabled if *LastIn* is low when *ReqIn* goes high and the second if *LastIn* is high (these annotations are made in the timing diagrams). *ReqIn* going high is used as the input-ready event for the receive (it allows for a 20ns setup time) and is determined by using the intervals of occurrence algorithm. The notation for the nodes is as follows: 0 implies a falling event, 1 is a rising event, X is a don't care, ! signifies that data is asserted, and ? signifies that the wire is tri-stated. Nodes are grouped if they occur simultaneously (within some tolerance).

APPENDIX 2

GeminiII: A Second Generation Layout Validation Program

Carl Ebeling
Department of Computer Science
University of Washington

ABSTRACT

Gemini is a circuit comparison program that is widely used to compare circuit layout against a specification. In this paper we describe recent extensions made to Gemini that make it faster, enable it to isolate errors better, and extend its domain of application. This has been done by changes to the labeling algorithm, extensions to the local matching algorithm, better handling of symmetrical circuits and the accommodation of series-connected transistors. GeminiII's algorithm is separated into global labeling and local matching phases. GeminiII dynamically switches between the two depending on the amount of local structure contained in the circuit, taking advantage of the speed of the local matching algorithm when possible and relying on the power of the more general algorithm when the simple algorithm fails. This blending of algorithms also allows differences between two circuits to be better contained so that defects can be pinpointed.

Introduction

One of the crucial steps in the design of VLSI circuits is that of determining whether the layout of the circuit geometry corresponds to the specification of the circuit. This problem is often addressed by simulating the circuit extracted from the layout. This method is both inefficient and ineffective. Simulation requires many hours for large systems and still is not guaranteed to find all errors. Moreover, subtle differences such as transistor sizing typically go undetected.

Gemini validates layout by comparing two circuits, the specification circuit and the circuit extracted from the layout. Gemini determines whether the two circuits match exactly, and if not, what the differences are. Gemini has been used to validate a wide range of chips, including the HITECH, CHIPTEST and SLAP chips at CMU, and the CRISP microprocessor at Bell Labs. The program is very fast, comparing circuits at the rate of about 300 transistors/second on a Sun-3 workstation. Thus even very large chips with 100,000 transistors take only a few minutes to compare.

The approach that Gemini takes is to treat the circuit comparison problem as an instance of the graph isomorphism problem. The circuits are represented as graphs and a heuristic algorithm based on *node invariants* is used to partition the graphs into groups of devices and nets which have the same characteristics. The partitioning is refined until each device or net is in a separate partition. At this point Gemini can match partitions between graphs to obtain the isomorphism mapping of the elements of one circuit onto the elements of the other.

This basic partitioning algorithm does very well at determining whether the circuits are identical. When circuits are different, however, Gemini attempts to isolate the differences. This is in

[1]This research was supported by the Defense Advanced Research Projects Agency, ARPA Contract Number MDA903-85-K-0072, and by NSF, Contract Number CCR-8657589.

general a difficult problem, since there are typically many different explanations for circuit differences. Gemini's approach is to match as many devices and nets as possible.

Gemini's graph matching algorithm is very powerful and efficient, achieving performance that is almost linear in the size of the circuit in most cases. However, some problems with Gemini became apparent over time. First, for some fairly simple circuits, Gemini was very slow to converge, in spite of 'obvious' structure in the circuits. This was especially apparent for highly symmetric circuits like register files and data paths. Second, Gemini was often unable to pinpoint the differences between circuits and offered the designer no help on how to proceed. Finally, Gemini did not allow series-connected transistors to be interchanged. In CMOS circuits where almost every complementary gate contains transistors in series, this became a serious drawback.

The improvements described in this paper address these problems. A new matching algorithm has been incorporated that uses local structure information to match nodes in the two graphs. Gemini uses this more efficient algorithm where possible, relying on the more general partitioning algorithm only when necessary. Gemini dynamically switches between the two algorithms based on the success of the weaker algorithm. A preprocessing step has also been added to allow the order of series-connected transistors to be ignored by the isomorphism algorithm.

The Partitioning *Algorithm*

Gemini determines whether two graphs are isomorphic using a partitioning algorithm[1, 2]. While the graph isomorphism problem appears to be very difficult in general[4], partitioning algorithms have been shown to be effective for a wide range of graph applications where pathological graphs do not occur in practice. Digital circuits fall in this category.

Circuits are represented in Gemini as bipartite graphs as shown in Figure 1, with devices and nets forming the two class of nodes. Devices are connected to nets through terminals which are divided into different classes. For example, the source and drain of a transistor are in the same terminal class since their connections can be interchanged without affecting the circuit. This very general way of representing circuits has several advantages. First, the representation is compact, using only $O(N)$ space for a set of N completely connected devices instead of $O(N^2)$ as required if only devices are represented. Second, this representation makes it convenient for the partitioning algorithm to alternate labeling the two classes of nodes. Finally, this representation is technology independent since it makes no assumptions about the devices or their interconnection. Gemini has been used for MOS and bipolar circuits for both digital and analog applications.

The partitioning of a graph is done implicitly by assigning a *label* to each node in the graph. This labeling is done using *node invariants*, which are properties that are maintained under an isomorphism mapping. Gemini uses the device type as the invariant for device nodes and the number of connections as the invariant for net nodes. An initial partitioning is done using these invariants as labels. This partitioning is refined by relabeling each node based on the labels of its neighbors, suitably modified by the terminal classes used to make the connections.

It is easy to show that if an isomorphism mapping exists between two graphs, then there is also a mapping between the partitions that are created by this labeling procedure. Moreover, the mapping can be easily derived from the labels. If all partitions contain a single node, then an isomorphism mapping has been found. The goal therefore is to relabel the nodes of the graphs until only singleton partitions remain.

Most circuits admit to this isomorphism algorithm and singleton partitions are in fact generated

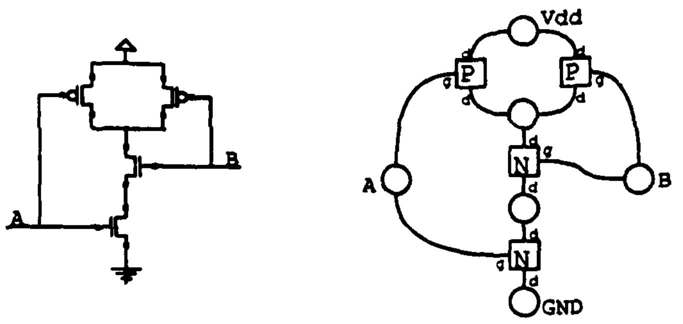


Figure 1: *A simple circuit and its graph representation. Devices are represented by squares and nets by circles. There are two terminal classes, drain/source represented by* d *and gate represented by* g.

by repeated labeling. Symmetric circuits, for which many distinct isomorphism mappings are possible, are an exception. Thus the partitioning algorithm cannot reduce the size of partitions that contain equivalent nodes. When Gemini detects symmetry, it arbitrarily matches nodes in corresponding partitions in the two circuits. Gemini's algorithm for doing this did not always work, often making only very slow progress and sometimes failing completely. GeminiII makes use of local matching as described in the next section to always make steady, if somewhat slow progress.

When a node is relabeled, all the information in the labels of neighboring nodes must be kept. This requires either progressively larger labels which are time-consuming to compute, or a renormalizing of labels after each step which is also time-consuming. Gemini instead uses approximate labels with values in the range $[0...N-1]$ with $N = 2^{32}$. Labels are then computed according to the hash function: $L^+ = \sum_i c_i L_i$ where L^+ is the new label, the L_i's are the labels of neighboring nodes and c_i is a factor that depends on the terminal class through which the neighbor is connected. It can be shown that with suitably chosen factors and initial labels, these approximate labels behave like the full labels with high probability.

Unfortunately, for very large circuits and circuits with a large effective diameter, collisions did in fact occur in Gemini resulting in non-singular partitions with differing nodes. These collisions occur because labels cannot maintain all information over a long number of relabelings. When this happened, Gemini was not able to reach any conclusion about the circuits. The labeling procedure was modified in GeminiII by assigning matching nodes new, unique labels as they are matched instead of relying on the unique labels generated by the labeling algorithm. This, in connection with the local matching algorithm, serves to minimize the loss of information in the approximate labeling.

Local Matching

The strength of Gemini's graph isomorphism algorithm is that it does not rely solely on local structure in the circuit to map the elements from one circuit to another. Many circuit comparison algorithms start from known nets such as inputs and progress from there by straightforward deduction about which nodes must match. This *local matching* approach works very well where circuits are locally structured. However, the kinds of large, regular circuits typically found in VLSI chips do not admit to this type of algorithm.

Gemini, by contrast, uses labeling to combine the local information over a large area until it finds nodes that must match. This algorithm can be inefficient, however, especially for large symmetric circuits since many nodes must be relabeled many times before being labeled uniquely.

GeminiII takes advantage of the strengths of the two different approaches by adding a simple, local matching algorithm to the general partitioning algorithm, using whichever is appropriate at each step of the matching process. The global partitioning algorithm is first used until some nodes are found to match. At this point, Gemini focuses on the neighbors of each pair of matching nodes and tries to match them based on strictly local information. This is done by partitioning just the neighbor nodes according to their values. These partitions must match in just the same way as the partitions of the graph itself and the nodes in singleton partitions can be immediately added to the isomorphism mapping. This local matching continues until there are no unprocessed matching nodes.

The two algorithms blend extremely well since they are cast in very similar terms. One examines partitions in the entire graph while the other examines partitions in a very small neighborhood. Thus most of the same data structures and procedures can be used for both algorithms. There is in fact a range of alternatives for local matching. An intermediate algorithm matches all the nodes in the local neighborhoods together instead of in each individual neighborhood. The performance of these alternatives are similar, but matching over restricted neighborhoods isolates circuit differences better.

The local matching algorithm does not discover anything that the global partitioning algorithm does not discover. However, it is more efficient since the work of global relabeling is not done. For most circuits over 95% of the labels can be found using local matching. However, the global partitioning algorithm is essential to provide the local algorithm 'anchor' points from which to start.

Isolating errors

The partitioning algorithm works extremely well when two circuits are the same, but it offers little help if the two circuits are different. In this case, the labels in the two graphs diverge rapidly and little useful information can be extracted. It is useful to think of one circuit as the standard and differences between the circuits as defects in the second. A single defect such as a switched or missing connection will cause all nodes within a distance D to be mislabeled after D relabelings. Since most circuits have a small diameter (typically less than 10), the entire circuit will be labeled differently after only a few relabelings. Gemini avoids this problem by matching all partitions after each relabeling. This allows singleton partitions to be matched as soon as possible and also allows non-matching partitions to be detected early and isolated so that differing labels do not propagate into the rest of the circuit.

Local matching also has a very beneficial effect on pinpointing defects. Matching is propagated without the global relabeling that causes large sections of the circuit to be mislabeled. This tends to localize the area affected by defects, and it also gives the designer a way to find defects when Gemini has trouble pinpointing them.

Gemini produces a node equivalence file as part of the matching process which can then be used as input to tell Gemini which nodes to match *a priori*. By starting out with matching nodes, the local matching algorithm can propagate matches into that part of the graph which contains the defect, narrowing down the area in which the problem occurs. Iterating this procedure typically narrows down the area containing the defect until it is isolated.

Symmetric Circuits

The running time of the partitioning algorithm increases dramatically for highly symmetric circuits. In these cases Gemini must make sure that partitions do in fact contain equivalent nodes before performing an arbitrary match. After every such match, Gemini continued with the labeling algorithm to allow partitions to subdivide if necessary. Many times, of course, this labeling did not accomplish anything.

GeminiII uses the local matching algorithm to quickly determine the result of arbitrarily matching two nodes in symmetric circuits. If no other nodes are affected, then no relabeling is required and the matching procedure can continue. This improved handling of symmetric circuits has dramatically improved the running time of GeminiII for symmetric circuits. For example, Gemini took 8 hours to compare one large circuit containing very fine-grained symmetry: GeminiII compares these in less than 5 minutes.

Series Transistors

Gemini compares circuits based on strict topological structure alone. Thus Gemini will consider two topologically different circuits different even if their functionality is the same. In the general case, it is extremely difficult to determine whether two topologically different circuits are functionally equivalent. However, series-connected transistors are a simple instance that is easy to recognize and that is particularly common in CMOS circuits.

The NOR gate of Figure 1 illustrates the problem with series transistors. Clearly the functionality of the circuit does not change if signals A and B are interchanged. However, the graph structure *is* changed and Gemini would report such a switch as an error. In practice, we have found that many designers want to know when such an interchange has been made since even though the circuit function remains the same, its performance can be affected substantially. For example, the size of transistors in a stack is typically increased towards the rail and late-changing signals are connected near the output. However, there are many designers who choose to ignore these issues, especially in CMOS designs where series transistors occur much more frequently than in NMOS.

The solution takes advantage of the fact that Gemini uses a general graph isomorphism algorithm that is oblivious to the actual types of devices contained in the circuit. A set of series-connected transistors, which we call a *chain*, can be thought of as a composite device that implements the AND switch function. That is, a chain comprising N transistors can be replaced with a pseudo-device with two source/drain terminals and N gate terminals as shown in Figure 2. The gate connections can now be permuted on this composite device without affecting the graph structure.

A pre-processing step that is part of the input procedure performs this graph reduction. As composite chain devices are matched, Gemini checks the order of the transistors and prints warnings if it differs in the two circuits. Isolating series transistors also serves to reduce the number of isomorphism errors, which can be more easily pinpointed by GeminiII.

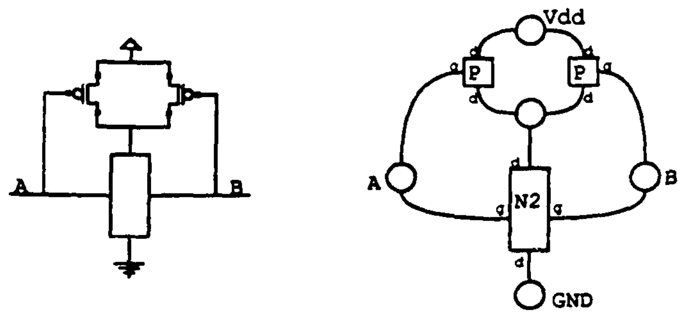


Figure 2: *The replacement of series-connected transistors by a composite device.*

Results

The graph in Figure 3 plots the running time for circuits of varying size and symmetry. The time measured is CPU time including I/O time for a Sun-3/180 workstation with local disk and 16Mbytes of memory. None of the circuits represented here suffered from substantial paging activity. We have chosen cases from the two extremes with respect to symmetry to indicate a performance envelope for all types of circuits.

For circuits with little or no symmetry, the running time is given by the equation $T = 2.6(\frac{N}{1000})^{1.09}$ where N is the number of transistors in the circuit. This is almost linear in the size of the circuit and extrapolating to a circuit with 1,000,000 transistors yields a running time of less than one hour.

The presence of symmetry increases the running time of GeminiII substantially. However, GeminiII is able to compare symmetrical circuits with predictable running times, in contrast to the original Gemini which often was unable to make a comparison or was very inefficient. The running time for highly symmetrical circuits is given by the equation $T = 3(\frac{N}{1000})^{1.85}$. This almost quadratic behavior is a result of the repeated scanning of partitions that Gemini performs when searching for the nodes most likely to be equivalent. The user can reduce the symmetry of the circuit by giving GeminiII a list of matching nodes such as inputs, outputs, and bus lines. However, large chips that contain components such as memory with a high degree of symmetry, usually contain interface circuitry that contains sufficient information to make the symmetry of the overall chip negligible.

Table 1 indicates the ability of GeminiII to find different kinds of defects. The ability of GeminiII to pinpoint errors is enhanced by the use of the local matching algorithm. If the difference between the two circuits is not substantial, then GeminiII typically pinpoints the error precisely. Sometimes, however, GeminiII will report nodes in error that are in fact correct. The usual procedure in this case is to rerun GeminiII while indicating which of the reported nodes in fact match. This forces GeminiII to find an alternative 'explanation' for the error. Depending on the number of errors, the user may have to repeat this procedure several times to find the location of the 'real' defect. In the limit, the correct explanation can be generated by prematching all nodes except those in error. The local matching algorithm allows one to approach this in effect by only prematching a few nodes in the vicinity of the reported error. These anchors are used to propagate matches, reducing the area in which errors are reported.

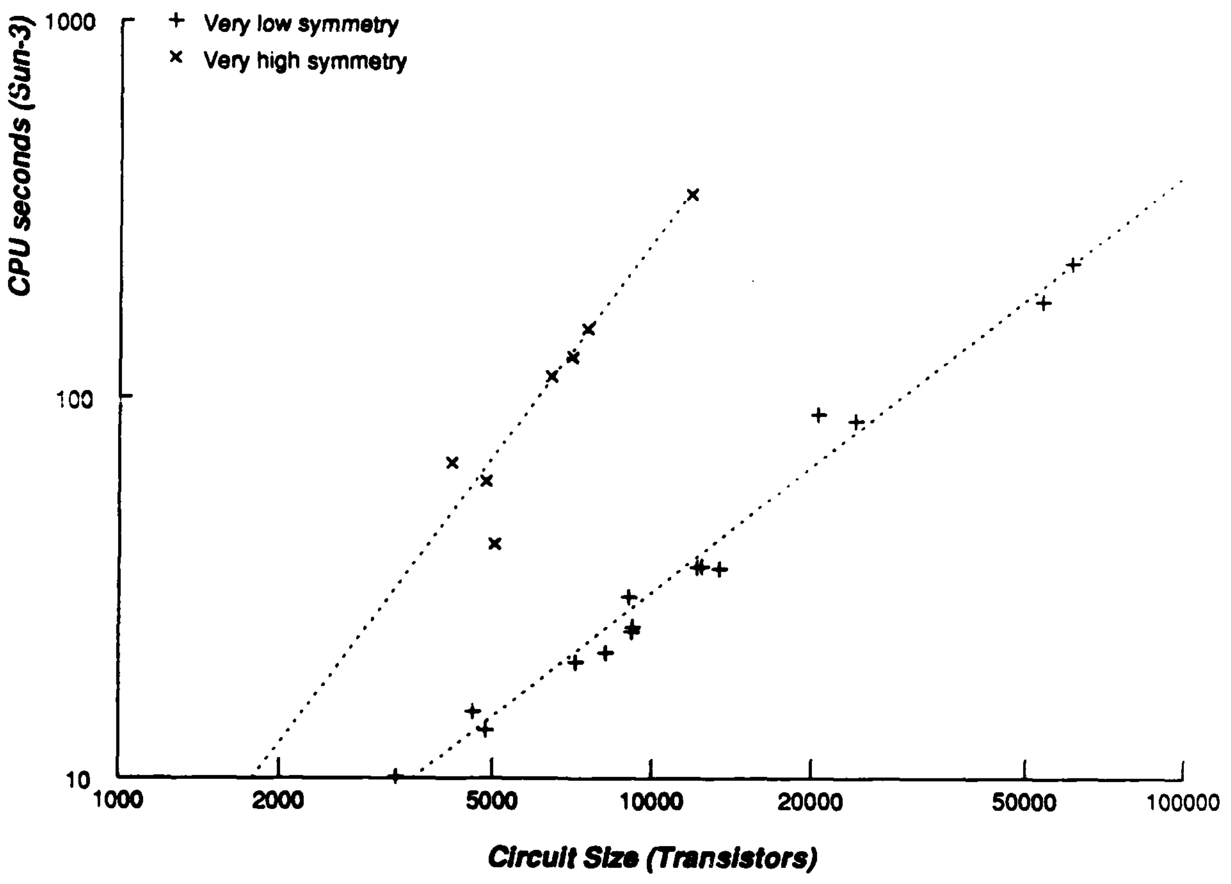


Figure 3: *GeminiII performance over circuits of varying size and symmetry.*

Type of error	CPU time	Spurious Errors reported
None	91	0
1 switched connection	116	0
Vdd/Gnd shorted	121	0
10 shorted connections	118	+5/-4
10 open connections	135	+2/-1
10 shorts, 10 opens	143	+10/-3
10 missing transistors	113	0
20 missing transistors	133	+3
100 missing transistors	181	+88
1000 missing transistors	450	+731

Table 1: *Performance of GeminiII when locating defects of various types. The circuit is a 25,000 transistor microprocessor with about 1% symmetry. Spurious errors indicates the number of non-defect nodes reported (+) and the number of defect nodes not reported (-).*

The type of error that continues to be the most difficult to analyze is that of shorted and switched connections. This type of defect affects many devices and nets unrelated to the cause of the error making it very difficult to isolate the defect in spite of the improvements described here. The better isolation of defects is the subject of continuing research.

Conclusion

GeminiII is a very efficient program for validating circuit layout against an independent specification. GeminiII retains the same general partitioning algorithm as Gemini, but makes better use of local matching to isolate errors and match symmetric circuits more efficiently. Collisions in the labeling hash function have been effectively eliminated by randomizing labels as nodes are matched. Finally, GeminiII recognizes permutations of series-connected transistors to be equivalent in function, increasing its utility for CMOS circuit designs.

References

[1] D. G. Corneil and C. C. Gotlieb. An algorithm for graph isomorphism. *Journal of the ACM*, 17:51–64, January 1970.

[2] D. G. Corneil and D. G. Kirkpatrick. A theoretical analysis of various heuristics for the graph isomorphism problem. *SIAM Journal of Computing*, 9:281–297, May 1980.

[3] C. Ebeling and O. Zajicek. Validating vlsi circuit layout by wirelist comparison. In *Proceedings of ICCAD*, pages 172–173, 1983.

[4] R. C. Read and D. G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1:339–363, 1977.

APPENDIX 3

The Influence of On-Chip Parallelism in the Performance of Event-Based Parallel Simulation *

Mary L. Bailey   Lawrence Snyder

Department of Computer Science
University of Washington
Seattle, WA 98195
November 8, 1988

1 Introduction

Circuit simulation is a bottleneck in VLSI design, and parallel computation has often been suggested as a means of speeding it up. But recent studies have shown very little parallelism available for exploitation in parallel simulation in VLSI chips [BS88, SB87, Fra85]. For example, in [BS88] the average parallelism, that is, the average number of nodes changing per timestep, for a 27,000 transistor CMOS chip was 6.4! Thus an infinite number of processors can only speed up this simulation by at most 6.4. This small amount of parallelism surprised chip designers who felt that their chips had more parallelism than these numbers showed and it challenged us to reconcile the apparent inconsistency. In this paper we:

> Explain the discrepancy between the small amount of parallelism available for exploitation in parallel simulation and the designer's perception that chips have much parallelism.

We do this by considering two metrics for measuring parallelism: the event metric used to measure the parallelism available for parallel simulation, and the queue metric which more closely represents the designer's idea of chip parallelism.

For parallel simulation, a common method for exploiting parallel processors is to partition the chip among multiple processors and execute the same algorithm on each portion of the chip. Achieving good speedup depends on keeping all of the processors busy. For synchronous algorithms, the event metric measures how busy the processors can be by computing the number of events that can be executed in parallel at each timestep assuming an unlimited

*This research is supported by the Defense Advanced Research Projects Agency, ARPA Contract Number MDA903-85-K-0072

number of processors and ignoring timesteps where no events occur. Thus, the event metric provides an upper bound on the speedup of synchronous parallel simulation. In [BS88] we used this metric to measure six chips, and found that the average parallelism ranged between 2.8 and 25, with the percentage of parallelism (the parallelism divided by the number of nodes in the circuit) ranging from 0.04% to 2.9%.

While the event metric provides the potential speedup for parallel simulation, it is really just averaging the number of node changes that complete simultaneously. But nodes do not change instantaneously, they charge or discharge over time. The simulation event is scheduled when the amount of charge in the node passes through a specific threshold. Thus, in the event metric two nodes are changing in parallel if and only if they reach their final values simultaneously.

Chip uesigners, however, consider two nodes to be changing in parallel if they are both in the *process* of changing state, *not* just whether they simultaneously reach their thresholds. Thus another metric is needed to capture this notion of parallelism. We use the queue metric which measures the average length of the simulation queue. The nodes on the queue are precisely those nodes that are in the process of changing state. Thus in the queue metric a node will contribute to the parallelism during *all* of the timesteps in which it is changing, as opposed to the event metric where it will contribute to the parallelism only at the timestep when it reaches its final value. We have measured nine chips using the queue metric and found that the parallelism ranges between 9.8 and 390, much larger values than have been observed with the event metric!

In [BS88] we also considered the effect of varying the timestep on the event parallelism measurements, and found that as the timestep increased, the parallelism also increased until it reached an upper limit, the parallelism found in unit delay simulations. This ceiling effect is due to the data dependencies in the chip - nodes that are dependent on a previous node must wait for the previous node to change before they can change. This is precisely the notion of unit delay; all nodes take one unit to change and all nodes affected by the changes in one time unit are evaluated in the next time unit. For the queue metric, varying the timestep has little effect on the resulting parallelism. However, the queue metric parallelism is also limited by the unit-delay parallelism, since in the unit-delay model a node is only on the queue for a single timestep. Thus as the timestep increases, the parallelism found using both metrics actually converge to that found using a unit-delay algorithm!

Hence, the reason for the discrepancy between the earlier measurements and the designer's perception is a difference in definition: whether a node contributes to the parallelism measurement for only one timestep or for multiple timesteps. Not all the activity on the chip can be effectively exploited using synchronous event-driven parallel simulation with fine-grained timesteps. However, at the cost of less precise timing, the parallelism found with the event metric rises and converges to the parallelism found using the queue metric.

2 Preliminaries

Now we present more formal definitions of the two metrics:

Definition 1 *The average parallelism reported by the event metric is the total number of events divided by the number of timesteps in which at least one event occurred.*

Definition 2 *The average parallelism reported by the queue metric is the sum of the queue lengths at each timestep in which there is at least one element in the queue divided by the number of such timesteps. Note here that when there are no more events on the queue, the simulation is finished.*

We used the RNL linear-level simulator developed by Terman [Ter83] for all of the measurements in this paper. It is similar to a switch-level simulator, but includes timing information as a part of its output. The default timebase for RNL is $0.1ns$. For the event metric measurements using the default timebase, we made no modifications to RNL, but used an output format which provides information on the events that RNL is processing. For the queue metric measurements, we modified RNL so that at the beginning of each timestep it printed out the size of its queue.

3 Circuit Parallelism

We have used the two metrics to measure the circuit parallelism on nine chips developed at the University of Washington. For all of the circuits we used extracted circuits for the simulations so that the actual topology of the circuits would be reflected in the measurements. We first describe the circuits, and then discuss the parallelism results.

1. The Apex I chip is a graphics co-processor chip that generates a large class of spline descriptions very quickly [DH87]. Its architecture is based on a triangular computation that generates points on a curve in a data-flow fashion. For the parallelism measurements, we initialized the triangle and used enough inputs to fill the pipeline and generate the first output. We then took parallelism measurements on the next 10 data points, which consists of three numbers, one for each of the x, y, and z coordinates.

2. The IIR digital filter was designed by Hyong Lee [Lee85]. It includes a 16×16 multiplier, a 32-bit ripple adder, a 9-bit ripple counter, a 17 stage, 16-bit shift register, four 3 stage, 16-bit shift registers, and a PLA. Here we measured one macrocycle containing 401 microcycles.

3. The Quarter Horse is a 32-bit RISC microprocessor [HJK*85]. It has 32 general purpose dual-ported registers, two internal busses, an ALU, shifter, memory address register, and a program counter structure with PLA control. In addition, it has an LSSD for testing purposes. As our test data we used a single run of a character load instruction, which takes 18 PLA cycles. The designers thought this instruction was highly parallel.

4. The PE (processor element) is a subcircuit of the Apex chip. The PE contains a 16×16 booth multiplier as well as a couple of adders. Three PEs are used in the APEX chip, so we took three sets of measurements, each consisting of generating the first output and measuring the parallelism of the 10 subsequent data points.

5. The Vertexgen is also a subcircuit of the Apex I chip. It consists of a 32-bit pipelined added with some associated registers and random logic. Three vertexgens are used in the APEX chip, so we took three sets of measurements for each subcircuit. Like the measurements of the chip, each set consisted of generating the first output and measuring the parallelism of the 10 subsequent data points.

The other four circuits are instances of generators, programs that produce families of circuits. For the generator instances, we averaged 20 sets of random inputs. Each data set consisted of enough random inputs to initialize the circuit followed by a random input for the parallelism measurement.

6. The Baugh-Wooley multiplier is a signed multiplier designed in static CMOS, and is purely combinatorial in nature [Sys87].
7. The Booth multiplier is a generator developed as a group project in a VLSI design class. Its design is based on the modified Booth multiplier using the sign generate method described in [Ann86]. The multiplier has a static CMOS multiplier plane and clocked pipeline registers between the multiplier plane and the final adder. An additional carry resolve unit is placed at each row in the multiplier plane to compute the carry generated by unnecessary low order bits. The final carry is then used as the *carryin* to the final 18-bit adder. The final adder is a precharged Manchester carry adder with carry bypass.
8. The shift register is a CMOS generator using two-phase non-overlapping clocks [Sys87]. Its latch is a master-slave dynamic latch implemented with two clocked inverters.
9. The decoder is a static gate style CMOS generator parameterized by the number of select lines [Sys87].

Table 1 shows the parallelism results using the queue metric. For each circuit we measured the average parallelism and the maximum parallelism. We then computed the percentage of parallelism by dividing the average parallelism by the number of nodes in the circuit. All of the values in the table are shown to two significant digits. The numbers in parentheses are standard deviations.

In general, the average parallelism decreases as the size of the circuit decreases. The two exceptions to this are the IIR Digital Filter and the Shift Register. The Shift Register exhibits more parallelism than many larger circuits, but was selected as a circuit which should exhibit large parallelism. The IIr Filter has less parallelism than might be expected, and in fact, has the lowest percentage of parallelism.

Even though the average parallelism generally decreases as the circuit size decreases, the percent parallelism doesn't have such nice properties. The percentage of parallelism ranges from 0.2% for the IIR Digital Filter to 8.9% for the Decoder. It is interesting to note that of the three biggest chips, only one has a percentage parallelism over 1, and this value is

Circuit	Transistors	Queue Parallelism (S.D.) Average		Queue Parallelism (S.D.) Percent		Max Queue Length	
Apex I	61,660	390	(150)	1.2%	(0.44)	1,800	(170)
IIR Digital Filter	27,360	31		0.22%		923	
Quarter Horse	24,068	100		0.95%		610	
PE	12,437	94	(18)	1.4%	(0.26)	470	(90)
Vertexgen	7,495	41	(8.6)	0.55%	(0.11)	220	(73)
8 × 8 Baugh-Wooley Multiplier	2,162	36	(9.9)	3.3%	(0.92)	98	(27)
8 × 8 Booth Multiplier	2,013	37	(7.0)	3.4%	(0.64)	120	(20)
8-Stage, 16-Bit Shift Register	1,536	91	(8.6)	8.7%	(0.82)	260	(24)
4 to 16 Decoder	208	9.8	(3.9)	8.9%	(3.6)	22	(10)

Table 1: Circuit Parallelism Using the Queue Metric

Circuit	Average Parallelism (S.D.)				Percent Parallelism (S.D.)			
	Event		Queue		Event		Queue	
Apex I	23	(2.1)	390	(150)	0.07%	(0.006)	1.2%	(0.44)
IIR Digital Filter	6.4		31		0.04%		0.22%	
Quarter Horse	6.3		100		0.06%		0.95%	
PE	8.0	(1.0)	94	(18)	0.12%	(0.015)	1.4%	(0.26)
Vertexgen	5.2	(1.0)	41	(8.6)	0.12%	(0025)	0.99%	(0.21)
8 × 8 Baugh-Wooley Multiplier	2.8	(0.50)	36	(9.9)	0.26%	(0.046)	3.3%	(0.92)
8 × 8 *Booth Multiplier*	3.4	(0.31)	37	(7.0)	0.31%	(0.031)	3.4%	(0.64)
8-Stage, 16-Bit Shift Register	25	(2.4)	91	(8.6)	2.4%	(0.23)	8.7%	(0.82)
4 to 16 Decoder	3.2	(0.47)	9.8	(3.9)	2.9%	0.091	8.9%	(3.6)

Table 2: Comparing Parallelism using the Events and Queue Metrics

not much greater than 1. Also the chip with the parallelism over 1, the Apex I chip, *is* a pipelined chip, so it was expected to hace more parallelism than the other chips.

Table 2 compares the parallelism of the event and queue metric. For each circuit we compare the average parallelism and the percentage of parallelism. The numbers for each metric are collected using the exact same input data for easy comparison. As expected, the parallelism measured using the queue metric is always larger than that measured using the event metric. However, the size of this difference is not constant. For these circuits, the queue parallelism ranges from being 3.1 to 17 times greater than the event parallelism. This is actually quite a large difference! Also the circuits with the largest percent parallelism using the event metric have the largest percent parallelism using the queue metric, although the difference. However, these two circuits were about an order of magnitude more parallel than the next most parallel circuits using the event metric, but only more parallel by a factor of 2.6 over the next most parallel circuits using the queue metric.

Thus, the parallelism is always greater using the queue metric, and it can be *much* higher for certain circuits. However, from these example circuits it is not clear that there is an obvious relationship between the two metrics. This is the topic for the next section.

4 Timestep Effects

When the timestep is increased, the parallelism using the event metric rises until it reaches the parallelism of unit-delay simulation [BS88]. This ceiling effect is due to the data dependencies in the circuit. Since in a unit-delay simulation, all of the events on the queue are evaluated in the next timestep, the definitions of event metric and queue metric are identical in this model. Thus, one would expect that if the timestep were increased, the parallelism in the queue metric would also find a ceiling at the parallelism using the unit-delay model.

Thus, we consider the effects of changing the timestep using the queue metric. The experiments were run using two circuits, a 16×16 instance of the Baugh-Wooley multiplier and a 6 to 64 instance of the decoder described in the previous section. To test the effect of timebase changes, we effectively modified RNL's internal timebase by changing the way queuing delays were calculated. For example, to change RNL's timebase from its internal timebase of $0.1ns$ to $n(0.1ns)$ we changed the delay Δd to:

$$\Delta d = \begin{cases} 0 & \text{if } \Delta d_{old} = 0 \\ n & \text{if } 0 < \Delta d_{old} < n \\ n(\Delta d_{old}/n) & \text{otherwise} \end{cases}$$

where "/" is integer divide. This insures that two dependent events do not occur in the same timestep. We also used a unit-delay option for RNL to provide measurements for the two metrics.

The results of these experiments are shown in Figures 1 and 2. We show the parallelism of both the queue and event metric for easy comparison.

For the Baugh-Wooley multiplier in Figure 1, the two metrics show quite different characters. For the event metric, the parallelism starts out quite small but grows rapidly until it reached the threshold of the unit delay timestep. The queue metric measurements start out quite large and increase only slightly. However, the surprise here is that the queue measurements may be higher than the unit delay measurements! We do not completely understand this effect, but it may be explained, at least partially, on the evaluation sequence imposed by the different algorithms. For example, depending on the evaluation sequence, certain nodes may oscillate more before settling to their final values. However, increasing the timestep increases parallelism up to a point, at which time the increase is negligible.

The Decoder has a more expected performance (Figure 2). Here the parallelism for the queue metric basically remains within that of the unit-delay computation for all timebases. The parallelism for the event metric rises for the first few timebases and then levels out as expected at the unit-delay level. Thus, in these two examples, the queue metric seems to generally reflect the unit-delay metric, while the event metric is much more sensitive to the timebase.

Another question related to the timebase is how different synchronization mechanisms affect the parallelism measurements. All of the measurements here are based on a synchronous model. Asynchronous computations allow different processors to be simultaneously executing events from different timesteps. This may increase the parallelism, at least for fine granularities of timebases.

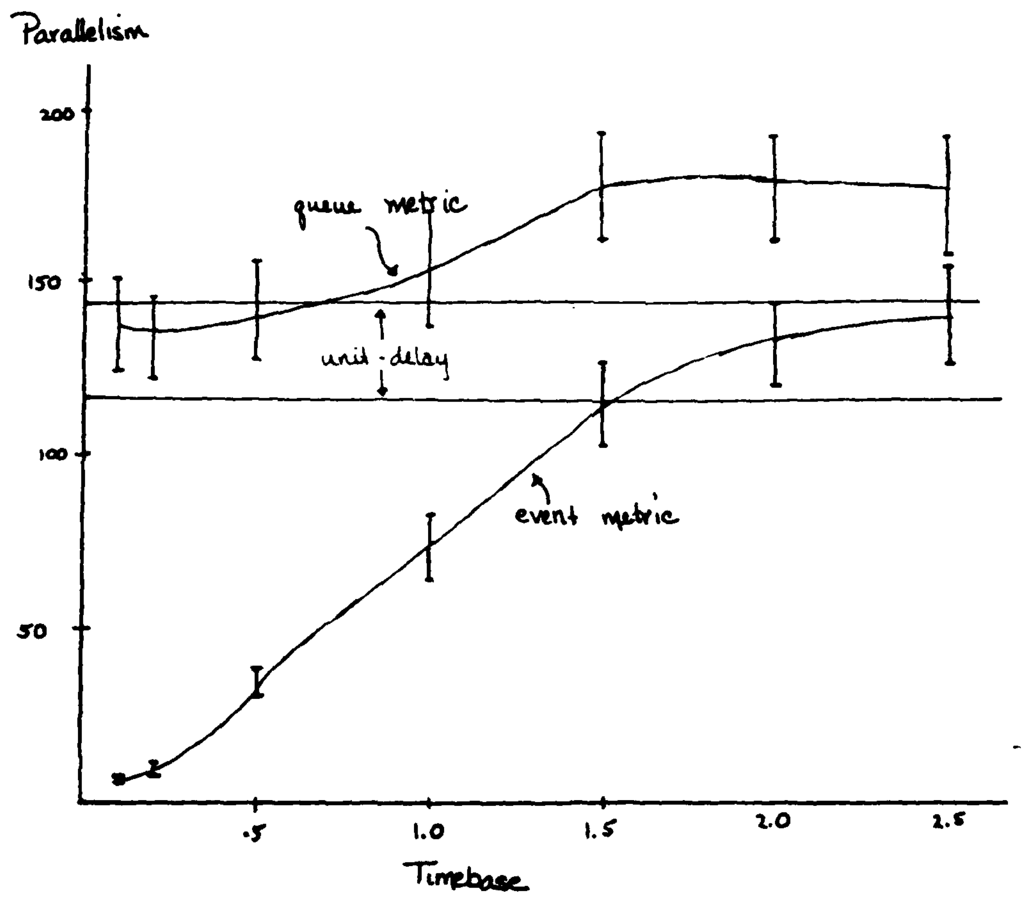


Figure 1: The Effect of Timebase in the Baugh-Wooley Multiplier.

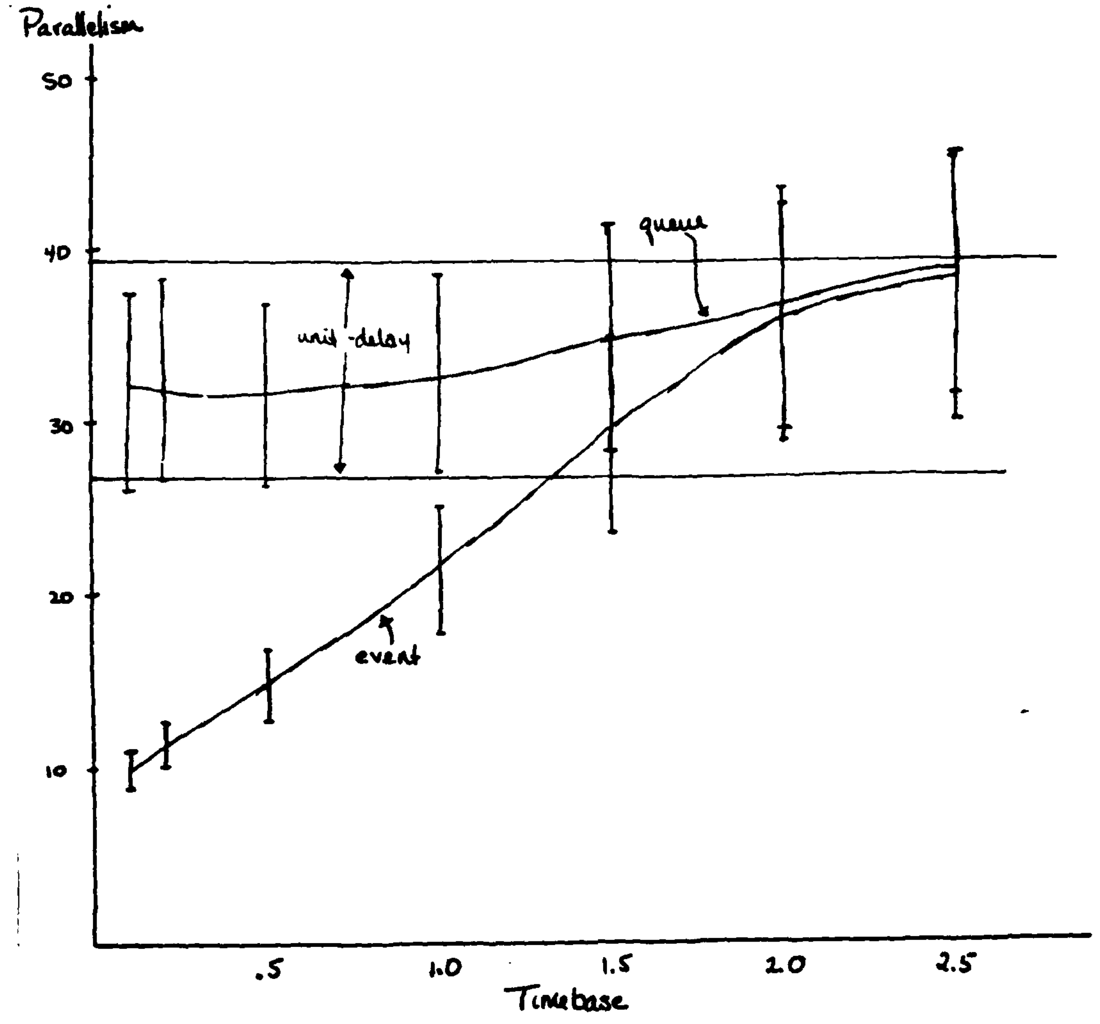


Figure 2: The Effect of Timebase in the Decoder.

5 Conclusions

In this paper we have answered the question: Why do chip designer's perceive more parallelism in their chips than has been found in the event metric. We have defined a new metric, the queue metric, which is analogous to the designer's idea of parallelism. We have measured nine chips using this metric and have compared the results to those found using the event metric. We found that the queue metric always finds more parallelism than the event metric, and can find up to a factor of 17 more parallelism.

Finally we have shown that by increasing the timebase the two parallelism found by the two metrics converge to that found using the unit-delay timing model. There are some fluctuations in the parallelism found using the unit-delay and the queue metric, but this may be at least partially explained by the different execution sequences that the algorithms exhibit. More experimentation needs to be done to insure that other effects are not present, or if they are, to identify them.

References

[Ann86] Marco Annaratone. *Digital CMOS Circuit Design*. Kluwer Academic Publishers, Norwell. Mass., 1986.

[BS87] Mary L. Bailey and Lawrence Snyder. *Measurements of On-Chip Parallelism in CMOS VLSI Circuits.* Technical Report TR87-11-03, University of Washington Department of Computer Science, 1987.

[BS88] Mary L. Bailey and Lawrence Syder. An Empirical Study of On-Chip Parallelism. In the *Proceedings of the 25th Design Automation Conference*, pp. 160-165. IEEE, June 1988.

[DH87] Tony D. DeRose and Thomas J. Holman. *The Triangle: A Multiprocessor Architecture for Fast Curve and Surface Generation.* Technical Report TR87-08-07, University of Washington Department of Computer Science, 1987.

[Fra85] Edward H. Frank. *A Data-Driven Multiprocessor for Switch-Level Simulation of VLSI Circuits.* PhD Thesis, Carnegie-Mellon University, November 1985.

[HJK*85] S. Ho, B. Jinks, T. Knight, J. Schaad, L. Snyder, A. Tyagi, and C. Yang. The Quarter Horse: A Case Study in Rapid Prototyping of a 32-bit Microprocessor Chip. In *Proceedings of the International Conference on Computer Design: VLSI in Computers*, pp. 261-266. IEEE, 1985.

[Lee85] Hyong Lee. *A Variable Digital Filter Design in 3um CMOS.* Master's Thesis, University of Washington, 1985.

[SB87] Larry Soule and Tom Blank. Statistics for Parallelism and Abstraction Level in Digital Simulation. In *Proceedings of the 24th Design Automation Conference* , pp. 588-591. IEEE, June, 1987.

[Sys87] Northwest Laboratory For Integrated Systems. *VLSI Design Tools Reference Manual Release 3.1.* Technical Report TR87-02-01, University of Washington Department of Computer Science, 1987.

[Ter83] Christopher J. Terman. *Simulation Tools for Digital LSI Design.* PhD Thesis, Massachusetts Institute of Technology, September 1983.